(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 398 492 B1

(12) EUROPEAN PATENT SPECIFICATION

(45) Date of publication and mention of the grant of the patent:
22.01.1997 Bulletin 1997/04

(51) Int Cl.[6]: **G06F 1/00**

(21) Application number: **90303877.6**

(22) Date of filing: **10.04.1990**

(54) A flexible interface to authentication services in a distributed data processing system

Flexible Schnittstelle für Beglaubigungsdienste in einem verteilten Datenverarbeitungssystem

Interface flexible pour les services d'authentification dans un système de traitement de données distribué

(84) Designated Contracting States:
DE FR GB IT

(30) Priority: **15.05.1989 US 352518**

(43) Date of publication of application:
22.11.1990 Bulletin 1990/47

(73) Proprietor: **International Business Machines Corporation**
Armonk, N.Y. 10504 (US)

(72) Inventors:
- **Loucks, Larry Keith**
  Austin,Texas 78750 (US)
- **Smith, Todd Allen**
  Austin, TX 78746 (US)

(74) Representative: **Bailey, Geoffrey Alan**
IBM United Kingdom Limited
Intellectual Property Department
Hursley Park
Winchester Hampshire SO21 2JN (GB)

(56) References cited:
EP-A- 0 268 141
US-A- 4 484 306
US-A- 4 694 492

- **"Kerberos: An Authentication Service for Open Network Systems", Steiner, Neuman, Schiller, pages 191-202, USENIX, Dallas, TX, Winter 1988.**

EP 0 398 492 B1

Printed by Jouve, 75001 PARIS (FR)

Description

This invention relates to a flexible interface to authentication sources in a distributed data processing system including a plurality of data processing systems connected by a communications link, and to the authentication of a process at one of the data processing systems for the use of a service at another one of the data processing systems in a distributed networking environment.

A portion of the disclosure of this patent document contains material which is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document or the patent disclosure, as it appears in the Patent and Trademark Office patent file or records, but otherwise reserves all copyright rights whatsoever.

As shown in Fig. 1, a distributed networking environment 1 consists of two or more nodes A, B, C, connected through a communication link or a network 3. The network 3 can be either a local area network (LAN), or a wide area network (WAN).

At any of the nodes A, B, C, there may be a processing system 10A, 10B, 10C, such as a workstation. Each of these processing systems 10A, 10B, 10C, may be a single user system or a multi-user system with the ability to use the network 3 to access files located at a remote node. For example, the processing system 10A at local node A, is able to access the files 5B, 5C at the remote nodes B, C, respectively.

Within this document, the term "server" will be used to indicate the processing system where the file is permanently stored, and the term "client" will be used to mean any other processing system having processes accessing the file. It is to be understood, however, that the term "server" does not mean a dedicated server as that term is used in some local area network systems. The distributed services system in which the invention is implemented is truly a distributed system supporting a wide variety of applications running at different nodes in the system which may access files located anywhere in the system.

As mentioned, the arrangement to be described hereinafter is directed to a distributed data processing system in a communication network. In this environment, each processor at a node in the network potentially may access all the files in the network no matter at which nodes the files may reside.

Other approaches to supporting a distributed data processing system are known. For example, IBM's Distributed Services for the AIX operating system is disclosed in S.N. 014,897 "A System and Method for Accessing Remote Files in a Distributed Networking Environment ", filed February 13, 1987 in the name of Johnson et al. In addition, Sun Microsystems has released a Network File System (NFS) and Bell Laboratories has developed a Remote File System (RFS). The Sun Microsystems NFS has been described in a series of publications including S.R. Kleiman, "Vnodes: An Architecture for Multiple File System Types in Sun UNIX", Conference Proceedings, USENIX 1986 Summer Technical Conference and Exhibition, pp. 238 to 247; Russel Sandberg et al., "Design and Implementation of the Sun Network Filesystem", Conference Proceedings, Usenix 1985, pp. 119 to 130; Dan Walsh et al., "Overview of the Sun Network File System", pp. 117 to 124; JoMei Chang, "Status Monitor Provides Network Locking Service for NFS", JoMei Chang, "SunNet", pp. 71 to 75; and Bradley Taylor, "Secure Networking in the Sun Environment", pp. 28 to 36. The AT&T RFS has also been described in a series of publications including Andrew P. Rifkin et al., "RFS Architectural Overview", USENIX Conference Proceedings, Atlanta, Georgia (June 1986), pp. 1 to 12; Richard Hamilton et al., "An Administrator's View of Remote File Sharing", pp. 1 to 9; Tom Houghton et al., "File Systems Switch", pp. 1 to 2; and David J. Olander et al., "A Framework for Networking in System V", pp. 1 to 8.

One feature of the distributed services system in which the disclosed arrangement is implemented which distinguishes it from the Sun Microsystems NFS, for example, is that Sun's approach was to design what is essentially a stateless server. This means that the server does not store any information about client nodes, including such information as which client nodes have a server file open or whether client processes have a file open in read_only or read_write modes. Such an implementation simplifies the design of the server because the server does not have to deal with error recovery situations which may arise when a client fails or goes off-line without properly informing the server that it is releasing its claim on server resources.

In contrast an entirely different approach was taken in the design of the distributed services system in which the disclosed arrangement is implemented which might be characterised as a "stateful implementation". A "stateful" server, such as that described here, does keep information about who is using its files and how the files are being used. This requires that the server have some way to detect the loss of contact with a client so that accumulated state information about that client can be discarded. The cache management strategies described here cannot be implemented unless the server keeps such state information.

The problems encountered in accessing remote nodes can be better understood by first examining how a stand-alone system accesses files. In a stand alone system, such as 10 as shown in Fig. 2, a local buffer 12 in the operating system 11 is used to buffer the data transferred between the permanent storage 2, such as a hard file or a disk in a workstation, and the user address space 14. The local buffer 12 in the operating system 11 is also referred to as a local cache or kernel buffer.

In the stand-alone system, the kernel buffer 12 is divided into blocks 15 which are identified by device number, and logical block number within the device. When a read system call 16 is issued, it is issued with a file descriptor of the file 5 for a byte range within the file 5, as shown in step 101, Fig. 3. The operating system 11 takes this information and converts it to device number, and logical block numbers in the device, step 102, Fig. 3. If the block is in the cache, step 103, the data is obtained directly from the cache, step 105. In the case where the cache doesn't hold the sought for block at step 103, the data is read into the cache in step 104 before proceeding with step 105 where the data is obtained from the cache.

Any data read from the disk 2 is kept in the cache block 15 until the cache block 15 is needed for some other purpose. Consequently, any successive read requests from an application 4 that is running on the processing system 10 for the same data previously read is accessed from the cache 12 and not the disk 2. Reading from the cache is far less time consuming than reading from the disk.

Similarly, data written from the application 4 is not saved immediately on the disk 2, but is written to the cache 12. This saves disk accesses if another write operation is issued to the same block. Modified data blocks in the cache 12 are saved on the disk 2 periodically.

Use of a cache in a stand-alone system that utilises an AIX operating system improves the overall performance of the system since disk accessing is eliminated for successive reads and writes. Overall performance is enhanced because accessing permanent storage is slower and more expensive than accessing a cache.

In a distributed environment, as shown in Fig. 1, there are two ways the processing system 10C in local node C could read the file 5A from node A. In one way, the processing system 10C could copy the whole file 5A, and then read it as if it were a local file 5C residing at node C. Reading a file in this way creates a problem if another processing system 10A at another node A modifies the file 5A after the file 5A has been copied at node C as file 5C. The processing system 10C would not have access to these latest modifications to the file 5A.

Another way for processing system 10C to access a file 5A at node A is to read one block, e.g. N1, at a time as the processing system at node C requires it. A problem with this method is that every read has to go across the network communication link 3 to the node A where the file resides. Sending the data for every successive read is time consuming.

Accessing files across a network presents two competing problems as illustrated above. One problem involves the time required to transmit data across the network for successive reads and writes. On the other hand, if the file data is stored in the node to reduce network traffic, the file integrity may be lost. For example, if one of the several nodes is also writing to the file, the other nodes accessing the file may not be accessing the latest updated data that has just been written. As such, the file integrity is lost since a node may be accessing incorrect and outdated files.

In a stand-alone data processing system, file access control is often provided in order to protect sensitive information that users do not want to share with each other. A problem confronted with distributed data processing systems is how to distribute this same model of control and provide secure access to a user's information remotely while keeping other remote users from inadvertently or maliciously accessing or manipulating data belonging to another user.

Two problems that must be solved in order to provide a secure remote access for users are authentication and authorisation. Authentication is the process of identifying a user of a data processing system. Users typically accomplish authentication by presenting the user's name, or account number for the system, followed by a secret password which should only be known by that user. Presenting the secret password, which can be validated by the system, allows the system to authenticate the user to be who the user claims to be. Once authenticated, the system may then authorise this user to have access to resources managed by the data processing system. Therefore, authentication is the identification of a user, and authorisation is the granting of a privilege to a user to gain some kind of access to the system.

As mentioned above, authentication of local users is often accomplished through the use of a shared secret. This secret is typically a password which the user enters at a prompt. The system then compares this password to a recorded version of the password. If the system determines that the password is correct, the user is authenticated. Remote authentication is more difficult than that previously described.

A procedure for remote authentication is described in the following papers. "Kerberos: An Authentication Service For Open Network Systems", Steiner, Jennifer G.; Neuman, Clifford; Schiller, Jeffrey I.; pages 1-15, USENIX, Dallas, TX, Winter, 1988. "Project Athena Technical Plan, Section E.2.1, Kerberos Authentication and Authorisation System", Miller, S.P.; Neuman, B.C.; Schiller, J.I.; Saltzer, J.H.; pages 1-36, Massachusetts Institute of Technology, October 27, 1988. This Kerberos based remote authentication authenticates users working at one node in a distributed data processing system to services running on the same node or other nodes in the distributed system. A distinctive property of the Kerberos protocol is that users can be authenticated with services running on nodes which share no secret with the user. If a simple password based authentication scheme were used, each user would have to share a secret with each machine in the entire distributed system. By using the Kerberos protocol, users need to share a secret with only one machine; the machine running the Kerberos authentication service.

Kerberos authentication works as follows. In order to be authenticated to a remote service, the user must present a specially constructed data structure, called a ticket, to the service that the user wishes to be authenticated to. This ticket is particular to the user and the service. That is, each service requires that these tickets are tickets intended for

that service. Moreover, these tickets are specially constructed for an individual user who wants to be authenticated. These tickets are issued by the part of the Kerberos authentication server called the ticket granting service. Before using the service, say a remote print server, a user acquires a ticket for that print server. The user requests the Kerberos ticket granting service to issue a ticket for that user for the use of the print server. The Kerberos ticket granting service constructs the ticket, gives it to the user, and the user presents it to the print server. The print server examines the ticket and can determine that the sender was indeed the claimed user.

In detail, this Kerberos scheme is more complex in order to ensure that tickets can not be forged or stolen by another user. In addition, the messages for passing tickets between machines are designed such that the messages can not be recorded and replayed. The Kerberos scheme also ensures that the user can not trick the ticket granting service itself. This last step can be accomplished by requiring a ticket to use the ticket granting service. This master ticket is acquired by users before any other ticket can be issued for that user. Once a user has a master ticket, the user can present the master ticket to the ticket granting service and be issued tickets for other services in the distributed system.

Another distinctive feature of the Kerberos authentication scheme is that while the user must go to the authentication server to request tickets, the service to which these tickets are being presented does not need to communicate with the Kerberos authentication service during the user authentication.

The Kerberos protocol provides a sophisticated authentication scheme. However, such schemes are not appropriate for small networks where the overhead in administering a Kerberos server may not justify its use. In such cases, other authentication schemes are possible which may just simply involve the sharing of a secret between every user and every node in the distributed system. If the distributed system has only a few machines, such as three or so, it may be easier for each user to maintain a password on each machine and to authenticate that password periodically to ensure that the users are not compromised.

A third possible authentication scheme might be appropriate for networks that are larger than those that can be conveniently managed by users needing individual passwords on each machine yet are smaller than would justify a Kerberos authentication protocol. This third scheme might use a centralised authentication server which all of the nodes communicate with. A password might be presented to a remote service by a user, and the remote service checks with the central authentication server to determine whether the password is correct. This requires services to communicate with the authentication server instead of users. This is in contrast to the Kerberos scheme where services do not need to communicate with the authentication server during user authentication.

As illustrated above, there is a wide range of possible implementations of authentication servers that may be appropriate for distributed systems providing user authentication.

An operating system, such as the AIX operating system, which provides for distributed functions such as in Distributed Services of the AIX operating system, uses remote authentication to authenticate client machines to servers, servers to client machines, and users to server machines. However, it may be desirable to run an operating system having distributed functions in various sizes of networks having a range of nodes from one or two nodes to hundreds or thousands of nodes.

Consequently, an operating system providing distributed functions must run in the presence of various authentication schemes from the simplest most straight forward scheme to the most complex and sophisticated scheme.

An efficient way of implementing a distributed services function of an operating system is to modify the operating system kernel to support distributed service operations. This includes the communication between nodes and the management and synchronisation of the facilities being distributed. When a user operation on one machine causes a request to be sent to a remote machine, that user must be authenticated to the remote machine. This activity causes programs to be executed at the remote machine. The distributed service function of the operating system must perform the authentication of the remote user, but to do so, it must use the network's authentication scheme, for which various schemes are possible and can operate very differently from one another. Additionally, this authentication may require communications with the remote authentication server at either the user side or at the remote machine side. Since it is impossible to anticipate all possible authentication schemes that the network may be using, it is desirable for a distributed service function of the operating system to have a flexible authentication protocol that allows it to use whatever available authentication scheme is running on the network.

For example, a distributed service of the operating system gets to a remote machine with a request from a user, but the remote machine may discover that it has never seen this user before. The remote machine then may require the user to authenticate itself to the remote machine. However, the distributed service function may not know how the authentication is to be performed. If the distributed service function determined how the authentication process is to be performed, this would limit the users of the distributed service function of an operating system to the one predetermined authentication scheme.

Furthermore, if the predetermined authentication scheme requires remote communications with the server, there is a variety of exceptions and failures which are difficult to provide for inside the kernel of the operating system at the point where the authentication process would need to be performed. If the communications to the authentication server

breaks, and the authentication process is inside the kernel, all processing within the data processing system may come to a halt while the operating system kernel is waiting while trying to communicate with the remote authentication server.

Accordingly, the present invention provides a system as defined in claim 1.

The present invention also provides a method of authenticating a requester, at a first node, of a service running at a second node according to claim 6.

An embodiment provides a method for use on a first data processing system having means for requesting, by a process at the first data processing system, a service at a second data processing system connected to the first data processing system through a means of communication, the method comprising: sending a message to a replaceable facility, supporting an authentication policy, separate from the requesting process, to initiate authentication by constructing authentication information; and sending, unaware of the contents of the constructed authentication information, the constructed authentication information, with a request to use the service, to the service for becoming authenticated to use the service.

A further embodiment provides a method for use on a second data processing system having means for receiving a request, from a process at the first data processing system, for a service, the second data processing system connected to the first data processing system through a means of communication, the method comprising: sending authentication information, received from the process, to a replaceable facility, supporting an authentication policy, separate from the service, for an interpretation of the authentication information and a construction of a plurality of authentication credentials; receiving the constructed credentials from the separate facility; performing an operation on the request based on the received credentials; and returning to the requesting process a result of the operation and the authentication acknowledgement received from the separate facility.

A still further embodiment provides a method for use with at least one data processing system, the method comprising: receiving a message from process to initiate authentication of the process in response to a request by the requesting process to use a service of a different process; constructing authentication information and an authentication acknowledgement from requester information and service information found in the received message; sending the constructed authentication information and the authentication acknowledgement to the requesting process; receiving the constructed authentication information from the service; constructing a plurality of credentials from the received authentication information for the service and a second authentication acknowledgement; and sending the constructed credentials and the authentication acknowledgement to the service.

The system and method disclosed hereinafter defines an interface between an operating system providing distributed service functions and a user program providing an authentication scheme. This interface is well defined and highly structured such that the kernel of the operating system does not change at all depending upon the authentication services in the network that are being utilised. A program providing an authentication scheme can be interchangeable with other authentication programs whenever the authentication scheme changes in the network. In addition, utilising a user program at the remote machine to perform the authentication allows the user program to be killed, restarted, and scheduled as a regular program without affecting the performance of the underlying operating system. The complexity of authentication lies in the program and not in the kernel of the operating system.

An authentication daemon program is used at both the user initiating node and the receiving service node. A set of message flows allows a distributed service function within the operating system to use the results of these authentication daemons in a way that is transparent to the distributed services function. The distributed services function merely passes the information back and forth without attempting to interpret the work of the authentication daemon. Accordingly, different authentication daemon programs using different authentication schemes can be used interchangeably within the network without affecting or changing the underlying distributed service function of the operating system. Likewise, the same distributed service function can be used within various networks having different authentication schemes in the authentication daemon used in each network.

More specifically, a request sent to a remote service that requires authentication includes an object called the authentication information object. This authentication information object and an associated authentication acknowledgement value are constructed, at the requester's node, by the authentication daemon from information about the requester. The contents of these authentication items and the means for their construction are preferably unknown outside of the authentication daemon itself. The authentication acknowledgement value is retained and the authentication information object is included in the message used for the request. The service, before performing the request, extracts the authentication information object from the request and passes it to a second authentication daemon running at the service's node. This second authentication daemon uses the authentication information object to authenticate the requester according to the authentication policy supported by the daemons. The second authentication daemon also returns a second authentication acknowledgement value that is returned to the requester in the reply to the request. This second authentication acknowledgement is compared with the first authentication acknowledgement previously retained by the requester to ensure that the identity of the remote service is that of the intended service.

The present invention will be described further by way of example with reference to an embodiment thereof and a contrasting prior art arrangement, both as illustrated in the accompanying drawings in which:

Fig. 1 is a block diagram of a distributed data processing system known in the art;

Fig. 2 is a block diagram showing a stand-alone data processing system known in the art for accessing a file through system calls;

Fig. 3 is a flow diagram of the data processing system of Fig. 2 accessing a file through a system call;

Fig. 4A is a request_for_service message which is used by a process running on a client machine in order to request that a remote service be performed;

Fig. 4B illustrates the messages that flow between the authentication agent and the requester and between the authentication agent and the service;

Fig. 5 shows a client data processing system and a server data processing system in a distributed data processing system having a separate authentication agent for performing authentication; and

Fig. 6 is a flow showing the operations of the requester, the authentication agent, and the service.

With reference to Figure 4A, the internode message request_for_service 410 used herein is described. The request_for_service message 410 is used by a process running on a client machine in order to request that a remote service be performed. The request 411 has an opcode 413 indicating the specific operation requested. The request 411 also has an operation field 420 used to indicate the desired operation to be performed by the server. The authentication info field 416 in the request is used to pass enough information to the remote machine to authenticate the process performing the request. The remote machine responds with the reply 412. The opcode field 414 indicates that this is the reply for the particular kind of request. The return code (rc) 417 in the reply is used to indicate the success or failure of the remote machines attempt to execute the request. An acknowledgement is returned in the ack field 419. The ack is used to verify that correct identification between the requesting process and the remote machine has occurred.

Fig. 4B shows the messages that flow between the authentication agent and the requester and between the authentication agent and the service.

The requester to authentication agent message 520 contains information describing the requester 531 and information describing the service 532 that the request will be sent to. The authentication agent to requester message 521 contains an authentication information object 416 and an authentication acknowledgement value 534. The service to authentication agent message 523 contains only the authentication info object 416. The authentication agent to service message 524 contains another authentication acknowledgement value 419 and a set of credentials 536 that are used by the service to authenticate the requester.

Referring to Fig. 5, a machine 503 contains a requester 504 that makes use of an authentication agent 502 through messages 520, 521. The authentication agent 502 refers to data stored on disk 501. A second machine 513 contains a service 514 that makes use of an authentication agent 512 through messages 523, 524. The requester 504 communicates with the service 514 via the request_for_service request 411 and its reply 412.

In the preferred embodiment, a first machine 503 is communicating with a second machine 513 over a network. Rather than performing the authentication operation within the requester process 504 and the service process 514, which may be running in an operating system of the machines, the authentication operation is performed in the authentication agent programs 502, 512 at both of the nodes. These authentication agents may be user level programs as used in systems running any type of operating system or daemons as used in systems executing the AIX operating system or other operating systems based on the UNIX operating system.

The following description refers to Fig. 4A, Fig. 4B, Fig. 5, and Fig. 6, concurrently. Before a process at a first machine begins to send a request 411, step 601, to use the services of a second machine, an authentication procedure must first be performed. The requesting process 504 sends a message 520 to the first authentication agent 502 at the first node in order to initiate authentication, step 602. This message contains information 531 describing the process making the request and information 532 identifying the requested service.

The first authentication agent 502 uses the contents of the message 520 to construct a reply 521, step 603, returned to the requesting process, step 604. This reply 521 contains authentication information 416 and an authentication ack 534. The way in which the authentication agent uses the contents of message 520 depends upon the particular authentication policy being supported; but the way that the requesting process uses the reply 521 is independent of the policy supported by the authentication agent. The requesting process does not interpret either the authentication information 416 or the authentication ack 534. The only operations that the requesting process needs to perform with these elements are the transmission of the authentication information to the remote service in a request, steps 605,

606 and a bitwise comparison for equality between the authentication ack 534 received and retained, step 606, from the local authentication agent 502, and the authentication ack 419 in reply 412 received from the remote service.

In addition, the service receiving the request 411 does not need to interpret the information contained in the authentication information field 416. The service, like the requester, treats the authentication information in the same manner independent of the authentication policy that it supports. The service always passes the authentication information to the authentication agent 512, step 607, where interpretation of this information is performed, step 608. The agent will find different contents in the authentication information message 523 depending upon the particular authentication policy being supported by the authentication agents 502 and 512. In the message 524 sent from the agent 512 to the service 514, step 609, the agent places a set of credentials 536 that describe the remote requester in a manner that is meaningful to the service. Additionally, the agent places an authentication ack 419 in this message that does not need to be interpreted by the service. Like the authentication information, the authentication ack 419 is treated in the same manner by the service for all authentication policies supported by the authentication agents. The only operation that the service performs on the authentication ack 419 is returning it to the requester in reply 412. The service makes a determination, step 610, based on the credentials 536 on the authentication and authorisation of the requester's permission to request an operation 420 of the service, conditionally performing the requested operation, step 611. The results of this determination and operation are returned in a return code 417 to the requester in a reply message 412 along with the authentication ack 419, step 612.

The requester receives the reply 412 and extracts the authentication ack 419 and compares it to the authentication ack 534 previously received from the local authentication agent 502, step 613. If the two acks are bitwise equal, the requester is assured of the remote service's identity to the limit of the ability of the authentication protocol supported by the agents. Upon verifying the identity of the service, the requester examines the return code 417 to determine the outcome of the original request, step 614, completing the request for service, step 615.

The following programming design language code illustrates the processing at the authentication agent.

```
/* authentication agent */
LOOP FOREVER
     await message;
     IF request is a requester to agent message THEN
          use requester information found in message
               along with the service information
               found in the message to construct both
               authentication information and
               an authentication acknowledgement;
          send message reply containing
               authentication information and
               authentication ack
               back to requester;
     ELSE /* request is a service to agent message */
          use the authentication info to construct a
               set of credentials and
               an authentication ack;
          send message reply containing
               the set of credentials and
               the authentication ack
               back to service;
     ENDIF;
ENDLOOP;
```


The following programming design language code illustrates the processing at the requester.

```
/* requester */

/*   about to make a request of a remote service */
construct a message containing a description of the
     process making the request and a description of
     the remote service;
```

```
send this message to authentication agent;
await reply from authentication agent;
save the authentication ack returned in reply from
     agent;
construct a request for remote service, include
     authentication information returned in reply
     from the authentication agent;
send request to remote service;
await reply from the service;
IF authentication ack in reply from service
     = saved authentication ack THEN
     /* remote service authenticated */
     examine the result of the remote operation;
ELSE
     /* remote service not authenticated */
     /* ignore the returned return code */
ENDIF;
```


The following programming design language code illustrates the processing at the service.

```
/* service */

LOOP FOREVER;
     await request from a remote requester;
     extract the authentication information from the
          request;
     send authentication information to the
          authentication agent;
     await reply from agent;
     extract credentials for remote requester from the
          reply;
     extract authentication ack from the reply and
          save it;
     IF credentials indicate that the requester has
          permission to have the server perform the
          operation requested in the request from the
          requester THEN
          perform the operation;
          return code := the results of the operation;
     ELSE
          return code := value indicating permission
               failure;
     ENDIF;
     construct a reply containing the return code and
          the saved authentication ack;
     send reply back to the server;
ENDLOOP;
```


Note, that the design suggested above employs messages to communicate between the service and the authentication agent and between the requester and the authentication agent. These messages are shown in Fig. 4B. This choice of communication interface is intended to allow multiple processes, either requesters or services or combinations of the two, running on the same node to use a single agent. In the preferred embodiment using the AIX operating system the interprocess communication facilities called message queues provide this type of interface. These message queue facilities are described in the "AIX Operating System Technical Reference", second edition, September, 1986, order number SV21-8009, part number 74X9990, and more specifically pages 2-71 to 2-85. This allows multiple processes to share the authentication daemon process. By using a well defined interface based on messages, the preferred embodiment allows the daemon process to be replaced without necessitating any changes in its clients, i.e., the requesters and the services.

For example, if Kerberos based authentication is being performed, the agent will acquire a Kerberos ticket for the requester to use the remote service. This may involve searching id to name tables stored in the password file on disk 501 or ticket caches stored on disk 501 or memory. It may further involve communication with a remote Kerberos server

using the Kerberos protocols. If Kerberos is used, the authentication info 416 contains all of the information that is needed by a service to authenticate a remote requester using Kerberos.

To illustrate how the preferred embodiment can be used to support the kerberos protocol, the steps performed by the authentication agent are further described. Message 520 to the authentication agent is used by the agent to determine the requesting process's associated user id. In the preferred embodiment, this id is contained in the field 531 describing the requester and is a small integer used to identify the user on whose behalf the request is running. The authentication agent uses this id to locate the Kerberos tickets owned by this user. In the preferred embodiment, the agent will find these tickets in a file located on the disk 501 under the name /tmp/tkt(uid) where (uid) is the actual user id. In this file, a ticket for the requested service described by field 532 may be found. If such a ticket is not found, a new ticket for the requested service is acquired by the agent. The agent acquires a new ticket by sending a ticket request to the Kerberos ticket granting service. This ticket request is constructed according to the requirements of the Kerberos protocol as described in "Kerberos Authentication and Authorisation System", Miller, S.P. et al. Basically, the agent constructs an encrypted authenticater containing a timestamp and sends it, along with a ticket for use of the ticket granting service, to the ticket granting service in its requests for a ticket to the desired service.

In response to the ticket request, the authentication agent will receive from the ticket granting service a ticket for the desired service and a session key for use with this service. The agent will also construct an authenticater encrypted with the session key an combine this with the ticket just received. The agent places this in the authentication info field 416 of the reply 521 that is returned to the requester. Additionally, the agent will construct an authentication ack from the value of the timestamp placed in the encrypted authenticater. It does this by adding one to the timestamp and encrypting it with the session key. This authentication ack is placed in field 534 of the reply.

The requester, upon receiving the reply 521, does not need to be aware that the authentication field and the authentication ack are constructed in order to take advantage of the Kerberos authentication protocols. Instead, the requester treats these as abstract values, handling them in the same way independent of the way in which the authentication agents are performing authentication. The authentication info field is sent to the remote service in message 411 where it is passed in message 523 to the authentication agent running at the remote node. The remote authentication agent obtains, from the ticket, verification of the remote requester's identity and the session key that will be used with this remote requester. This information is encrypted in the ticket in a manner which makes tickets unforegable. Additionally, the remote agent checks the authenticater sent with the ticket, decrypting it with the session key, to find the timestamp. The timestamp is examined to make sure that this request is not a replay of a previous request. The timestamp is also used to construct an authentication ack in the same manner as was done by the first authentication agent. The authentication ack is returned to the service in field 419 along with a description of the requester's identity in field 536 of message 524. The ack constructed at the service authentication agent is then returned in message 412 to the requester where it is compared to the authentication ack constructed at the requester's node. This completes a Kerberos based authentication scheme using the preferred embodiment.

While performing the above described request, the agent communicates with the Kerberos ticket granting service. This may be running on a remote node and hence require network communication. This is not a problem for the agent since it is a regular scheduled process which can be put to sleep while awaiting a reply from the Kerberos ticket granting service without adversely impacting the performance of other processes on the system where the agent is running. If the requester had attempted to acquire the tickets directly without going through an authentication agent, the requester might have to sleep awaiting an I/O request. This could impact system performance adversely if the requester was running under kernel state in some operating systems. Furthermore, some operating systems cannot initiate I/O such as this while in the middle of processing a previous I/O request for a process. This can occur if a process attempts to perform I/O on a remote file. While in the middle of an I/O request for a remote file, it may be discovered that authentication operations need to be performed that involve I/O such as communicating with the Kerberos ticket granting service, causing problems in systems where a separate authentication agent is not used. In this preferred embodiment, authentication operations can be performed by the separate agent while a requesting process is in a state where it would not be able to perform the I/O necessary for the authentication at that time.

Likewise, a different authentication policy can be used in conjunction with this preferred embodiment. When the authentication agent receives message 520 it obtains the user id from field 531. It should be noted that in the preferred embodiment all information that might be used by an authentication policy is passed in field 531 so that different implementations of a policy or different policies running in the agent will find all information needed in this field. The user id is used to search the password file on disk 501 for a password that this user has prearranged to use with the remote service identified by field 532. This password and an associated reply password are obtained by this lookup operation. The password is placed in field 416, the authentication info field, and the reply password is placed in field 534, the authentication ack field, of the message 521. The requester sends the authentication info field 416 to the service where it is passed to the remote agent in message 523. The remote agent uses the password to search a file stored on disk 511 to locate a description of the identity of the requester. If found, this identity information is placed in field 536 along with a reply password found in the same file and placed in field 419, the authentication ack field of message 524. The

service returns the authentication ack value to the requester where it, the reply password found at the service node, is compared with the authentication ack obtained from the requester authentication agent, the reply password found in the local file. If they are the same, successful mutual authentication has occurred.

Another possible authentication policy might be implemented with a centralised authentication system. Like the scheme just described, the authentication info field is a password passed from the requester to the service and checked in the service authentication agent. The authentication ack is a reply password passed to the requester from both the requester authentication agent and the service, which obtains it from the service authentication agent. Instead of finding these passwords in files on disks 501, 511, these passwords are obtained from a centralised authentication system by the agent.

Finally, the agents may be in direct communication with each other. This allows authentication policies that work outside of the network used for request for service operations to be supported. For example, a physically secured communication channel between the agents could be used to dynamically determine appropriate authentication info values and authentication ack values. Requesters and services can communicate over lower cost or more available communication channels while achieving secure authentication through the use of the preferred embodiment where the agents communicate over a single secure channel.

It should be noted that the process acting as the authentication agent in a node can act in both roles, as the requester authentication agent and the service authentication agent because both services and requesters may be running in the same machine.

Simply by disconnecting the authentication daemons, and replacing them with another authentication daemon program, the network can perform authentication using any authentication scheme. For example, the authentication daemon could perform either a centralised authentication scheme or a Kerberos authentication scheme. Alternatively, the authentication daemon may simply just examine a file containing passwords.

As shown above, the authentication of remote users by servers can be performed in various ways. Under some circumstances, there is very little reason to be suspicious of requests arriving at a server and low cost authentication of remote users is justified. In other environments, servers must exercise greater vigilance. No one policy will be best for all cases. Therefore, this disclosed arrangement supports a range of authentication and authorisation policies.

Claims

1. A system having means for authenticating a requester (504) running at a first node (503), of a service (514) running at a second node (513), wherein the requester (504) and the service (514) are connected by a means of communication (411;412) in a distributed data processing system, the system comprising:

   means for constructing authentication information (416) and a first authentication acknowledgement (534) by a first facility, supporting an authentication policy, running at the first node separate from the requester (504);

   means for sending, by the requester (504), the authentication information to the service (514) running at the second node (513);

   means for processing the sent authentication information at a second facility, supporting the authentication policy, running at the second node (513) separate from the service;

   means for acquiring, by the service, an outcome of the processing for determining an authentication (536) of the requester (504) and a second authentication acknowledgement (419), and means for sending the second authentication acknowledgement to the service;

   means for receiving, by the requester (504), the second authentication acknowledgement; and

   means for comparing, by the requester (504), the first authentication acknowledgemet (534) and the second received authentication acknowledgement (419) for determining an authentication of the service (514).

2. A system as claimed in claim 1, further comprising means for replacing the first and second separate facilities supporting the authentication policy with different separate facilities supporting a different implementation of the authentication policy.

3. A system as claimed in any preceding claim, wherein the requester (504) is a process running for a user on an operating system at the first node (503) and the separate facility is a daemon process.

4. A system as claimed in claim 3 wherein the daemon process performs operations, to construct the authentication information, that the requesting process is incapable of performing at the time of the request.

5. A system as claimed in any of claims 1 to 4, wherein the requester is a process running on an operating system at the first node and the separate facility is accessible by at least one other process running on the operating system.

6. A method of authenticating a requester running at a first node, of a service running at a second node, wherein the requester and the service are connected by a means of communication in a distributed data processing system, the method comprising:

constructing authentication information and a first authentication acknowledgement by a first facility, supporting an authentication policy, running at the first node separate from the requester;

sending, by the requester, the authentication information to the service running at the second node;

processing the sent authentication information at a second facility, supporting the authentication policy, running at the second node separate from the service;

acquiring, by the service, an outcome of the processing for determining an authentication of the requester and a second authentication acknowledgement and sending the second authentication acknowledgement to the service;

receiving, by the requester, the second authentication acknowledgement; and

comparing, by the requester, the first authentication acknowledgement and the second received authentication acknowledgement for determining an authentication of the service.

7. A method as claimed in claim 6, further comprising replacing the first separate facility and the second separate facility with different separate facilities supporting a different authentication policy.

Patentansprüche

1. Ein System mit einem Mittel zur Beglaubigung eines Anforderers (504), der an einem ersten Knoten (503) läuft und einen Dienst (514) an einem zweiten Knoten (513) anfordert, bei dem der Anforderer (504) und der Dienst (514) über eine Übermittlungsstrecke (411; 412) in einem verteilten Datenverarbeitungssystem miteinander verbunden sind, wobei das System folgendes umfaßt:

Mittel zum Aufbau von Beglaubigungsinformationen (416) und einer ersten Beglaubigungsbestätigung (534) durch eine erste Einrichtung, die eine an dem ersten Knoten separat vom Anforderer (504) laufende Beglaubigungsmethode unterstützt;

Mittel zum Senden der Beglaubigungsinformation vom Anforderer (504) an den Dienst (514), der an dem zweiten Knoten (513) läuft;

Mittel zum Verarbeiten der gesendeten Beglaubigungsinformation an einer zweiten Einrichtung, welche die Beglaubigungsmethode unterstützt, und an dem zweiten Knoten (513) getrennt von dem Dienst läuft;

Mittel, mit dem der Dienst ein Ergebnis der Verarbeitung zur Feststellung einer Beglaubigung (536) des Anforderers (504) und eine zweite Beglaubigungsbestätigung (419) erwirbt, und Mittel, um die zweite Beglaubigungsbestätigung an den Dienst zu senden;

Mittel, mit dem der Anforderer (504) die zweite Beglaubigungsbestätigung empfängt; und

Mittel, mit denen der Anforderer (504) die erste Beglaubigungsbestätigung (534) und die zweite empfangene Beglaubigungsbestätigung (419) vergleicht, um eine Beglaubigung des Dienstes (514) festzustellen.

2. Ein System nach Anspruch 1, weiter umfassend Mittel, mit denen die erste und die zweite separate Einrichtung,

welche die Beglaubigungsmethode unterstützen, gegen andere separate Einrichtungen ausgetauscht werden, die eine andere Implementierung der Beglaubigungsmethode unterstützen.

3. Ein System nach einem jeden vorangehenden Anspruch, bei dem der Anforderer (504) ein Prozeß ist, der für einen Benutzer auf einem Betriebssystemniveau an dem ersten Knoten (503) läuft und bei dem die separate Einrichtung ein Dämonprozeß ist.

4. Ein System nach Anspruch 3, bei dem der Dämonprozeß zum Aufbau der Beglaubigungsinformation Operationen ausführt, die der anfordernde Prozeß zum Zeitpunkt der Anforderung nicht ausführen kann.

5. Ein System nach einem jeden der Ansprüche 1 bis 4, bei dem der Anforderer ein Prozeß ist, der auf einer Betriebssystemebene an dem ersten Knoten läuft, und bei dem auf die separate Einrichtung von mindestens einem anderen Prozeß zugegriffen werden kann, der auf Betriebssystemebene läuft.

6. Eine Methode zur Beglaubigung eines Anforderers, der an einem ersten Knoten läuft, und der einen Dienst anfordert, der an einem zweiten Knoten läuft, bei der der Anforderer und der Dienst über ein Kommunikationsmittel in einem verteilten Datenverarbeitungssystem miteinander verbunden sind, wobei die Methode folgendes umfaßt:

Aufbauen einer Beglaubigungsinformation und einer ersten Beglaubigungsbestätigung durch eine erste Einrichtung, die eine Beglaubigungsmethode unterstützt, und die an dem ersten Knoten getrennt vom Anforderer läuft;

Senden der Beglaubigungsinformation an den an dem zweiten Knoten laufenden Dienst durch den Anforderer;

Verarbeiten der gesendeten Beglaubigungsinformation an einer zweiten Einrichtung, welche die Beglaubigungsmethode unterstützt, und die an dem zweiten Knoten getrennt von dem Dienst läuft;

Erwerben eines Ergebnisses der Verarbeitung zur Feststellung einer Beglaubigung des Anforderers und einer zweiten Beglaubigungsbestätigung durch den Dienst und Senden der zweiten Beglaubigungsbestätigung an den Dienst;

Empfangen der zweiten Beglaubigungsbestätigung vom Anforderer; und

Vergleichen der ersten Beglaubigungsbestätigung und der zweiten empfangenen Beglaubigungsbestätigung zur Feststellung einer Beglaubigung des Dienstes durch den Anforderer.

7. Eine Methode nach Anspruch 6, weiter umfassend das Austauschen der ersten getrennten Einrichtung und der zweiten getrennten Einrichtung gegen andere getrennte Einrichtungen, die eine andere Beglaubigungsmethode unterstützen.

Revendications

1. Système ayant un moyen pour authentifier un demandeur (504) fonctionnant à un premier noeud (503), d'un service (514) exécuté à un deuxième noeud (513), où le demandeur (504) et le service (514) sont reliés par un moyen de communication (411; 412) dans un système de traitement des données réparti, le système comprenant:

un moyen pour qu'un premier dispositif, supportant une politique d'authentification, fonctionnant au premier noeud séparément de l'élément demandeur (504), construise des données d'authentification (416) et un premier accusé de réception de l'authentification (534);

un moyen pour que le demandeur (504) envoie les données d'authentification au service (514) exécuté au deuxième noeud (513);

un moyen pour traiter les données d'authentification reçues vers un deuxième dispositif, supportant la politique d'authentification, et fonctionnant sur le deuxième noeud (513) séparé du service;

un moyen pour que le service obtienne le résultat du traitement pour déterminer une première authentification

(536) de l'élément demandeur (504) et un deuxième accusé de réception d'authentification (419), et un moyen pour envoyer au service le deuxième accusé de réception de l'authentification;

un moyen pour que l'élément demandeur (504) reçoive le deuxième accusé de réception de l'authentification; et

un moyen pour que l'élément demandeur (504) compare le premier accusé de réception d'authentification (534) et le deuxième accusé de réception d'authentification (419) pour déterminer une authentification du service (514).

2. Système tel que revendiqué dans la revendication 1, comprenant en outre un moyen pour remplacer les premier et deuxième dispositifs séparés qui supportent la politique d'authentification par des dispositifs séparés différents supportant une mise en application différente de la politique d'authentification.

3. Système selon l'une quelconque des revendications précédentes, où l'élément demandeur (504) est une procédure exécutée pour un utilisateur à un niveau du système d'exploitation sur le premier noeud (503) et où le dispositif séparé est un processus "daemon" (démon).

4. Système tel que revendiqué dans la revendication 3 où le processus "daemon" exécute des opérations, pour construire les données d'authentification, que la procédure demandeuse est incapable d'exécuter au moment de la requête.

5. Système selon l'une quelconque des revendications 1 à 4, où le demandeur est une procédure exécutée à un niveau du système d'exploitation sur le premier noeud, et où au moins une autre procédure exécutée sur niveau du système d'exploitation peut accéder au dispositif séparé.

6. Méthode d'authentification du demandeur fonctionnant sur un premier noeud, d'un service fonctionnant sur un deuxième noeud, où le demandeur et le service sont reliés par un moyen de communication dans un système de traitement réparti des données, la méthode comprenant les phases qui consistent à:

faire construire des données d'authentification et un premier accusé de réception de l'authentification par un premier dispositif, supportant une politique d'authentification, fonctionnant au premier noeud, et distinct du demandeur;

faire envoyer, par le demandeur, les données d'authentification au service exécuté sur le deuxième noeud;

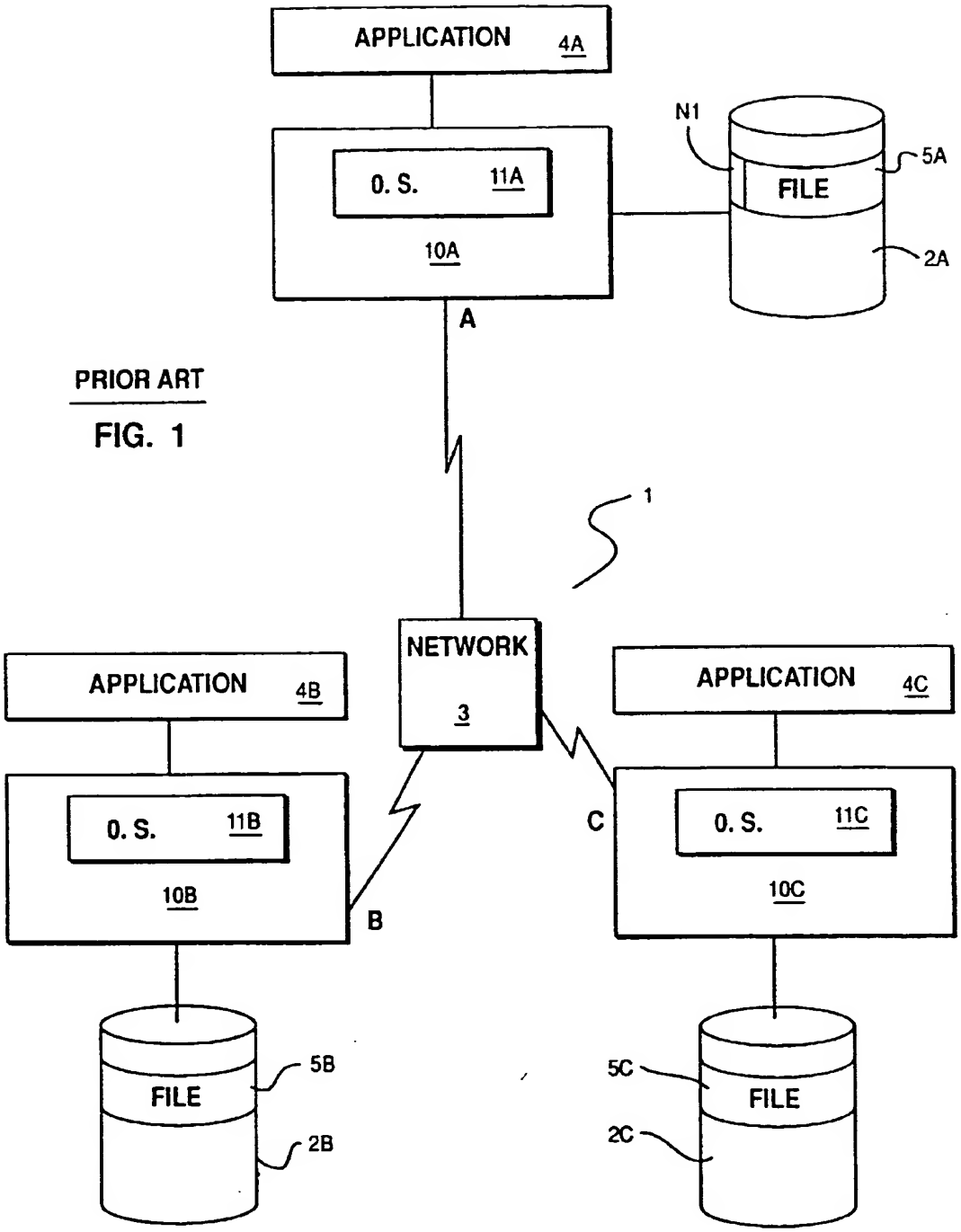
traiter les données d'authentification reçues dans un deuxième dispositif, supportant la politique d'authentification, et fonctionnant sur le deuxième noeud, séparément du service;

obtenir, pour le service, le résultat du traitement pour déterminer une authentification du demandeur et un deuxième accusé de réception de l'authentification et envoyer le deuxième accusé de réception de l'authentification au service;

faire recevoir, par le demandeur, le deuxième accusé de réception de l'authentification; et

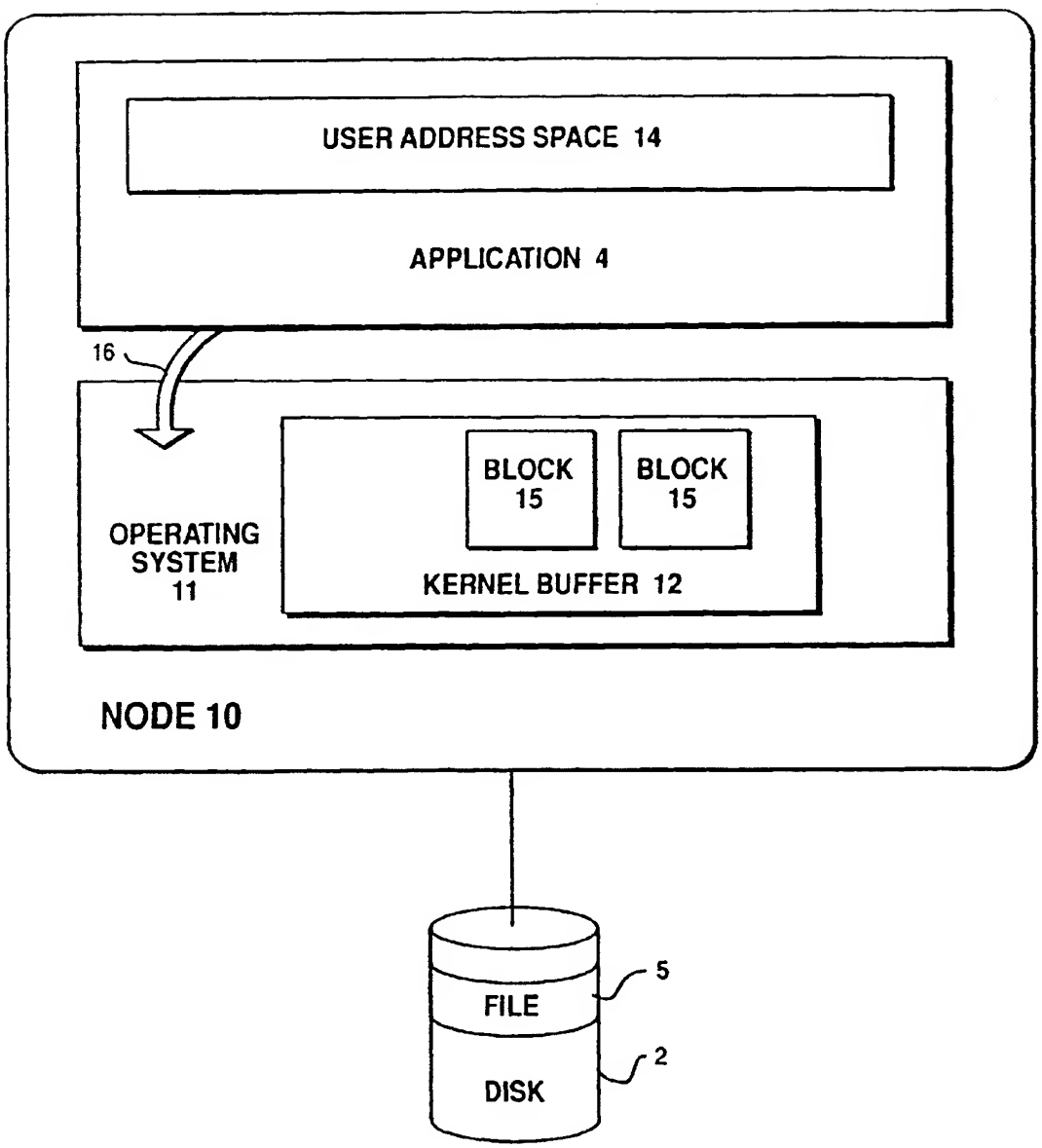
faire comparer, par le demandeur, le premier accusé de réception de l'authentification et le deuxième accusé de réception de l'authentification reçu pour déterminer une authentification du service.

7. Méthode telle que revendiquée dans la revendication 6, comprenant en outre le remplacement du premier dispositif séparé et du deuxième dispositif séparé par des dispositifs séparés différents supportant une politique d'authentification différente.



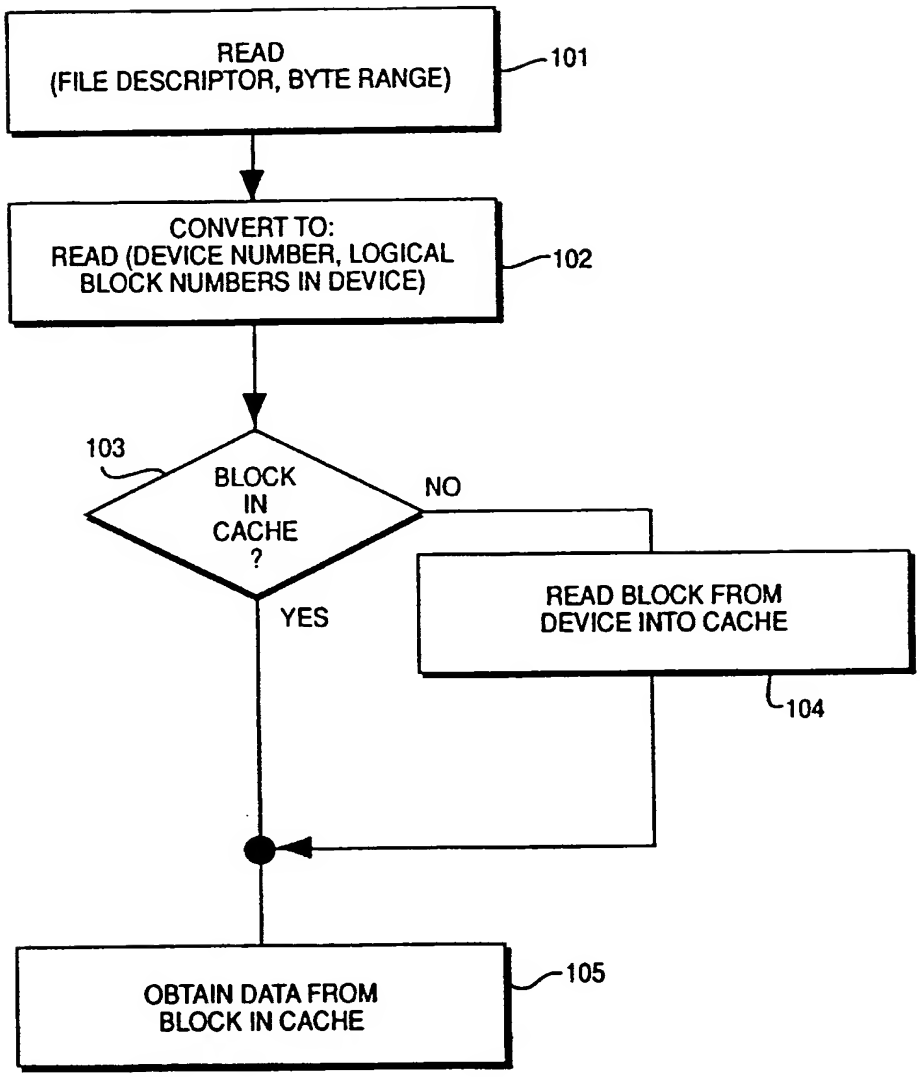
PRIOR ART

FIG. 1



PRIOR ART

FIG. 2



PRIOR ART

FIG. 3

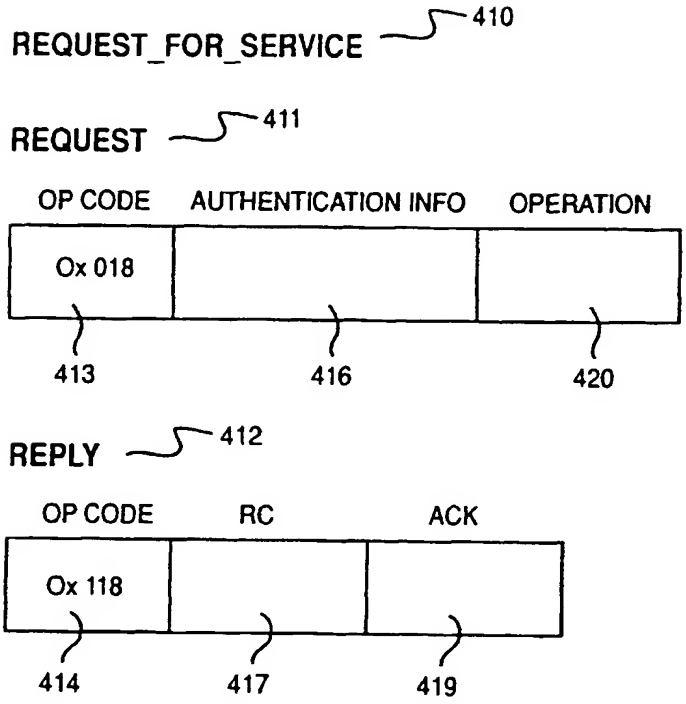
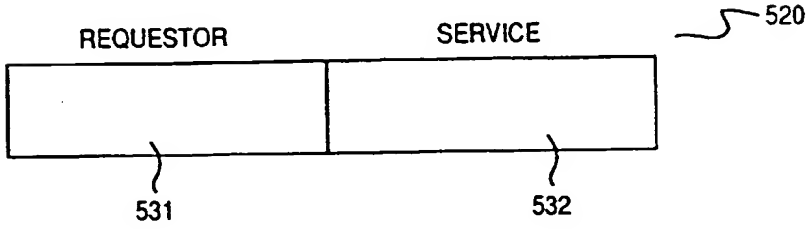
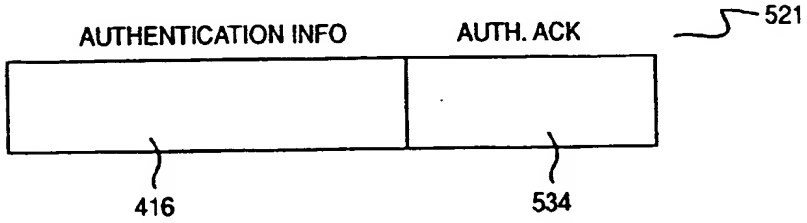

REQUEST_FOR_SERVICE
410
REQUEST
411
OP CODE
AUTHENTICATION INFO
OPERATION
Ox 018
413
416
420
REPLY
412
OP CODE
RC
ACK
Ox 118
414
417
419

FIG. 4A

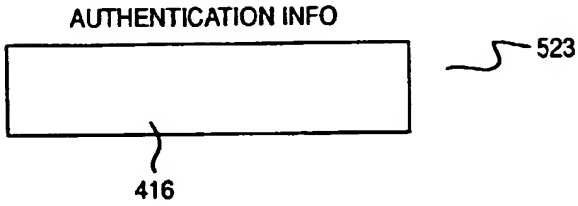
REQUESTOR TO AUTHENTICATION AGENT MESSAGE



AUTHENTICATION AGENT TO REQUESTOR MESSAGE



SERVICE TO AUTHENTICATION AGENT MESSAGE



AUTHENTICATION AGENT TO SERVICE MESSAGE

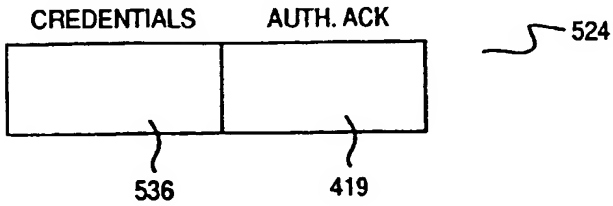


FIG. 4B

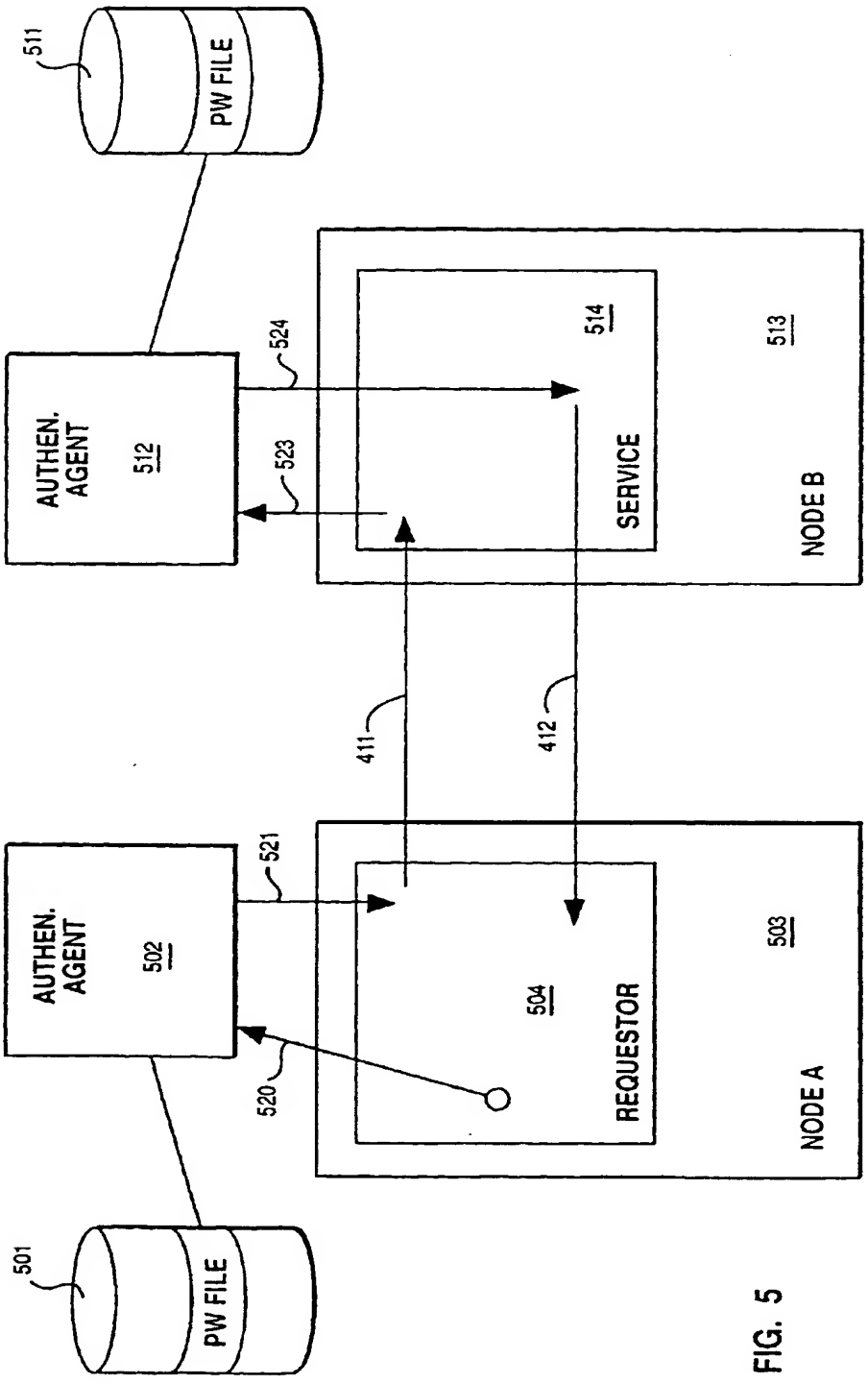

511
PW FILE
AUTHEN. AGENT
512
524
523
514
SERVICE
513
NODE B
411
412
AUTHEN. AGENT
502
521
520
504
REQUESTOR
503
NODE A
501
PW FILE

FIG. 5

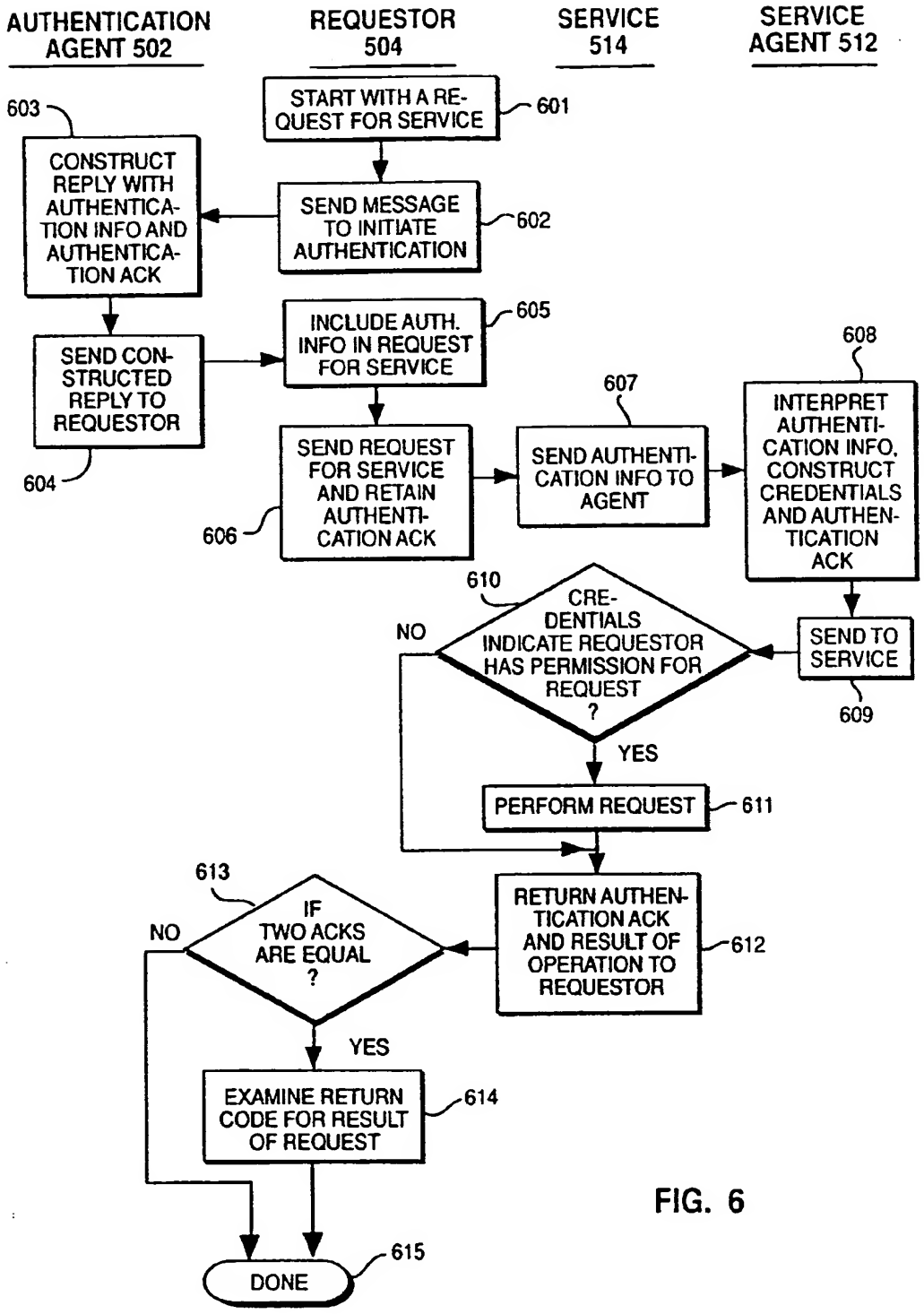

AUTHENTICATION AGENT 502
REQUESTOR 504
SERVICE 514
SERVICE AGENT 512
START WITH A REQUEST FOR SERVICE
601
SEND MESSAGE TO INITIATE AUTHENTICATION
602
CONSTRUCT REPLY WITH AUTHENTICATION INFO AND AUTHENTICATION ACK
603
SEND CONSTRUCTED REPLY TO REQUESTOR
604
INCLUDE AUTH. INFO IN REQUEST FOR SERVICE
605
SEND REQUEST FOR SERVICE AND RETAIN AUTHENTICATION ACK
606
SEND AUTHENTICATION INFO TO AGENT
607
INTERPRET AUTHENTICATION INFO, CONSTRUCT CREDENTIALS AND AUTHENTICATION ACK
608
SEND TO SERVICE
609
CREDENTIALS INDICATE REQUESTOR HAS PERMISSION FOR REQUEST ?
610
NO
YES
PERFORM REQUEST
611
RETURN AUTHENTICATION ACK AND RESULT OF OPERATION TO REQUESTOR
612
IF TWO ACKS ARE EQUAL ?
613
NO
YES
EXAMINE RETURN CODE FOR RESULT OF REQUEST
614
DONE
615

FIG. 6